

---

# **Grid User Management System v.0.7.1**

**Project Documentation**

---



# Table of Contents

---

<b>1</b>	<b>Manual</b>	
1.1	Introduction .....	1
1.2	How to: quick installation .....	4
1.3	Installation guide .....	9
1.4	Configuration .....	15
1.4.1	gums.config .....	16
1.4.2	db.properties .....	23
1.4.3	hosts.conf .....	24
1.5	Commands: GUMS Admin .....	25
1.6	Logging .....	27
1.6.1	Logging implementation .....	29
1.7	Integration .....	31
1.8	Operation modes .....	34
1.9	FAQ .....	36
1.10	Changes .....	39



## 1.1 Introduction

---

### What does GUMS provide?

*In this article we describe what GUMS is, what functionalities is going to provide to a site and how does it fit in the GRID architecture. We'll also give a brief history of GUMS. The reader by the end will have a general idea of GUMS and will be able to decide whether or not the tool is a match for their needs.*

### What GUMS is

GUMS performs one and just one function: mapping grid certificates to local identities (i.e. UNIX account). For example, a job comes to a site with a GRID credential (the PROXY certificate). The site resource, the gatekeeper, will need to map the GRID credential to the site credential under which the job will actually be running. GUMS is a service that provide that mapping: tells you which site user the GRID user should be using.

Notice that it doesn't authenticate for you: it doesn't 'su', it doesn't retrieve Kerberos credentials. It just tells the gatekeeper which site credentials should get. The gatekeeper is still in charge of enforcing the site mapping established by GUMS. Technically speaking, it's a Policy Decision Point (PDP) not a Policy Enforcement Point (PEP). GUMS provide a web service door to generate grid-mapfile or to map users one at a time. The first is used by a command line tool to retrieve grid-mapfiles that can be installed on each gatekeeper as a cron job; the latter is used by a gatekeeper callout module which is used instead of the grid-mapfile. Though the first tool is provided by the "GUMS Host tools" package, we refer to GUMS as the service that maintains the mapping.

GUMS interface for the callout is being implemented according to standards discussed at GGF. The existence of this interface means that any kind of service is able to contact GUMS: even though we are working on a GT2 gatekeeper callout, the same service can be used for a GT4 service, or a different type of service altogether.

### Mapping in GUMS

The mappings in GUMS are defined by a single XML policy file. Within the policy the administrator can specify which policy to use to map different groups of users, and on which hosts to use the mapping. For example, one can specify "on 'atlas\*.bnl.org' I want to allow all the USATLAS users, taken from the ATLAS VOMS server, and map them to a group account named 'usatlas01'; on those same hosts I also want to allow all the ATLAS users mapped to accounts taken from a site pool of accounts".

GUMS allows to be extended to meet the specific site services and requirements. All GUMS policy components (i.e. user group definition, mapping policies, ...) are implementation of a few simple interfaces, which implementation is not coupled to GUMS. This means that, with very little knowledge of GUMS itself, a site admin can write an extra piece of code that, for example, performs the mapping

according to the information present on a site service, or decide to keep GUMS data stored in their LDAP system.

The components already implemented in GUMS allow you to:

- Retrieve membership information from VO server such as LDAP and VOMS.
- Keep a manual group definition, useful to handle special cases
- Map groups of users to the same account (Group account)
- Map groups of users to an account pool, one account will be assigned to each user
- Map groups of users according to the information present in NIS
- Map groups of users according to a manual mapping definition, useful to handle special cases

### **The development of GUMS**

GUMS was first designed by Rich Baker and Dantong Yu at BNL in the first half of 2003. A first implementation was provided by Tomasz Wlodek and Dantong Yu. Gabriele Carcassi took over the project in March 2004 and brought the system into full production at BNL in May 2004. Between June and July the code was refactored to allow the business logic to be called either from command line or from a web application, opening the door to a web service implementation.

Current work is going toward a web application that would provide both a web interface for the administrator and a web service that implement the OGSA AuthZ interface. This is done within the Privilege Project, a joint project between USCMS and USATLAS.

### **GUMS in the future**

The need to map to local identities might, in the future, go away. The grid community, and especially the security groups, are pushing toward a model in which a GRID job would only be able to access any other service through GRID credentials. Also, a job would not be allowed to leave any trace of its passage. In fact, all traces might be wiped out. With that in mind, the particular user to which the job would be mapped would become less important. Right now, though, many authorizations decisions are still done through the username and uid.

The following is an incomplete list of authorizations that will need to be addressed before local account mapping becomes irrelevant:

- Access control on files. If file access for all inputs and outputs of a job would go through an interface with grid authorization, say an SRM, then whether a job was mapped to a particular account wouldn't affect file access privileges. As long as we use directly UNIX file systems for storage, local account mappings are crucial.
- Priority on a batch system. Many batch systems use uids and gids to determine submission rights to queues or to determine priorities across users. Batch system interfaces that makes decisions only on GRID credentials would be needed.
- Identify running processes. The easiest way to track process running on a host is to look at the username under which is running. A mechanism would be needed to easily go back from process id to a GRID identity

As long as those needs are there, there will be a need to keep a mapping, and there will be a need to keep

that mapping consistent and under control across the site. GUMS will be there to satisfy that need.

**Comparison with other tools**

Some people might get confused between GUMS, VOMS, SAZ, LCMAPS and other siteAAA tools. We keep a brief comparison within our FAQ.

## 1.2 How to: quick installation

---

### HOW TO: GUMS Service quick-start installation

*This article describes how to quickly install a GUMS service, without going through many of the details.*

#### Prepare the database

The first to do is to prepare the database. There is currently no script to do it: it's something we plan on doing.

You will need a mysql server, with version 4.0.18 or greater installed. You can either install one from scratch (follow the instruction on mysql's site) or you can use an installation you have ready. Log in as root and run the following script (cut and paste from this page):

```
CREATE DATABASE GUMS_0_7;

GRANT ALL
  ON GUMS_0_7.*
  TO gums@'<gumsserver.site.gov>' IDENTIFIED BY '<password>';

USE GUMS_0_7;

CREATE TABLE User (
  userID INTEGER AUTO_INCREMENT PRIMARY KEY,
  userDN VARCHAR(255) NOT NULL,
  userFQAN VARCHAR(255) DEFAULT NULL,
  registrationDate DATETIME NOT NULL,
  removalDate DATETIME DEFAULT NULL,
  userGroup VARCHAR(255) NOT NULL
);

CREATE TABLE UserAccountMapping (
  mappingID INTEGER NOT NULL,
  userDN VARCHAR(255) DEFAULT NULL,
  account VARCHAR(31) NOT NULL,
  startDate DATETIME DEFAULT NULL,
  endDate DATETIME DEFAULT NULL,
  userGroup VARCHAR(255) NOT NULL
);

CREATE TABLE HostGridMapfile (
  mapfileID INTEGER AUTO_INCREMENT PRIMARY KEY,
  hostname VARCHAR(255) NOT NULL,
  mapFile MEDIUMTEXT NOT NULL
);

CREATE TABLE HostGrid3UserVoMap (
  mapfileID INTEGER AUTO_INCREMENT PRIMARY KEY,
  hostname VARCHAR(255) NOT NULL,
  mapFile MEDIUMTEXT NOT NULL
```



```
);
```

in which you should replace `<gumsserver.site.gov>` with the machine on which you will run the GUMS server component, and choose a `<password>`.

You should also add your certificate in the admin group:

```
INSERT INTO User SET userDN="/DC=org/DC=doegrids/OU=People/CN=Gabriele Carcassi",
userGroup="admins";
```

### Preparing java

GUMS is written in java, and requires java to be installed to run. Be sure it is installed in your \$PATH. Try running:

```
[root@gums root]# java -version
java version "1.4.2_04"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.2_04-b05)
Java HotSpot(TM) Client VM (build 1.4.2_04-b05, mixed mode)
```

If you do not have java installed, go to <http://java.sun.com> and follow the instructions to get the latest version. Then add java to \$PATH.

### Preparing the certificate directory

GUMS will need the GRID certificate for the host and the certificate directories in place.

TODO: Explain how to setup this using VDT.

### Preparing Tomcat + EGEE security

The GUMS service will require a web server container, configured to use SSL with Globus proxy certificates. You will also need Xerces 2.5.0 in the common/endorsed directory. If you do not know what that means, just grab the already packaged Tomcat from the download page and install it.

- Grab the tarball from the download section [root@gums root]# mkdir /opt/gums-service  
[root@gums root]# cd /opt/gums-service [root@gums gums-service]# wget  
<http://grid.racf.bnl.gov/GUMS/dist/tomcat5.0.28EGEEsec.tar.gz> .
- Untar it [root@gums gums-service]# tar -xvzf tomcat5.0.28EGEEsec.tar.gz
- Review the configuration of the server [root@gums gums-service]# vi conf/server.xml

NOTE: if you need to change the configuration for the service certificate, or the port on which the service runs, you can edit the \$TOMCAT\_HOME/conf/server.xml tomcat configuration file. Find the section:

```
<Connector port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
```

```
enableLookups="false" disableUploadTimeout="true" acceptCount="100" debug="0"
scheme="https" secure="true"
sSLImplementation="org.glite.security.trustmanager.tomcat.TMSSLImplementation"
sslCAFiles="/etc/grid-security/certificates/*.0" crlFiles="/etc/grid-security/certificates/*.r0"
sslCertFile="/etc/grid-security/hostcert.pem" sslKey="/etc/grid-security/hostkey.pem"
log4jConfFile="/opt/gums-service/conf/log4j-trustmanager.properties" clientAuth="true"
sslProtocol="TLS" />
```

If you installed in a different directory than /opt/gums-service, change the location of log4jConfFile (TODO can this be made relative to tomcat.base?). To change the location of the service certificate or the CAs, simply change the sslXxx and crlFiles properties. To change the port, change the port property.

- Start the server [root@gums gums-service]# bin/catalina.sh start
- Connect to the server through a web browser with a Grid certificate installed, to check that is indeed running.

## Install the service

The service itself is a standard java web application. Grab the latest gums-service-0.7.1dev-latest.war file from the [dist directory](#), and unpack it in your \$TOMCAT\_HOME/webapps/gums directory. Tomcat will auto-mount.

- Grab the latest build and install [root@gums root]# cd /opt/gums-service/ [root@gums gums-service]# wget http://grid.racf.bnl.gov/GUMS/dist/gums-service-0.7.1dev-latest.war . [root@gums gums-service]# mkdir webapps/gums [root@gums gums-service]# cd webapps/gums [root@gums gums]# jar -xf ../gums-service-0.7dev-build<latest>.war
- Check the configuration file for the database information: [root@gums gums]# vi WEB-INF/classes/gums.config You will see something like: <persistenceFactory name='mysql' className='gov.bnl.gums.MySQLPersistenceFactory' jdbcDriver='com.mysql.jdbc.Driver' jdbcUrl='jdbc:mysql://\*your\_server\*/GUMS\_0\_7' user='gums' password='\*password\*' autoReconnect='true' /> You will need to set the server and password. If you need to pass extra parameters to the MySQL database driver, you can simply add them as attributes. For example, autoReconnect is a property for the driver to retry the connection if it dropped.
- Restart the tomcat server: [root@services01 gums]# ../bin/catalina.sh stop Using CATALINA\_BASE: /opt/gums-service Using CATALINA\_HOME: /opt/gums-service Using CATALINA\_TMPDIR: /opt/gums-service/temp Using JAVA\_HOME: /usr/java/j2sdk [root@services01 gums]# ../bin/catalina.sh start Using CATALINA\_BASE: /opt/gums-service Using CATALINA\_HOME: /opt/gums-service Using CATALINA\_TMPDIR: /opt/gums-service/temp Using JAVA\_HOME: /usr/java/j2sdk
- Get a browser in which you have your grid certificate, go to: <https://<machine>:<port>/gums> : you should see the GUMS web interface.
- Try generating the mapfile for the host "testing.site.com", and it should give you some response.
- [Optional] Another thing you can do is activate the e-mail forwarding of the log in case of error. Edit /opt/gums-service/webapps/gums/WEB-INF/classes/log4j.properties, and you will see a commented out section. Fill in the appropriate information, restart the service, and whenever GUMS will encounter an error, an e-mail will be sent to the address you specified.

- [Optional] GUMS provides a log suitable for cybersecurity in the `/opt/gums-service/logs/gums-site-admin.log`. The same log can be configured to use syslogd. Please refer to the logging documentation for more details.
- [Optional] One of the things GUMS does, is downloading the information from the VO servers every 12 hours. The clock will start at the first access to GUMS functionalities after each restart (i.e. first time you actually generate a mapfile or map a user). The time between updates can be set. The easiest way to do it, is to edit the `/opt/gums-service/webapps/gums/WEB-INF/web.xml` file.
 

```
<env-entry> <env-entry-name>updateGroupsMinutes</env-entry-name>
<env-entry-type>java.lang.Integer</env-entry-type> <env-entry-value>720</env-entry-value>
</env-entry>
```

 Change the entry value to the number of minutes you prefer.

Congratulations: the server is up and running.

## Installing the admin client

NOTE: GUMS Admin doesn't need to be installed on the same machine than GUMS. But it needs to be installed on a machine where `grid-proxy-init` and your user certificate are available.

Now you will need to install GUMS admin to access the command line tools. Grab the latest `gums-admin-0.7.1dev-latest.noarch.rpm` file from the [dist directory](#) : it will install by default in `/opt/gums`, but the package is relocatable, so you can install it wherever you want.

- Grab the latest build and install `[root@gums root]# wget http://grid.racf.bnl.gov/GUMS/dist/gums-admin-0.7.1dev-latest.noarch.rpm . [root@gums root]# rpm -Uvh gums-admin-0.7dev-build<latest>.noarch.rpm`
- Check that GUMS was installed correctly `[root@gums root]# cd /opt/gums/bin/ [root@gums bin]# ./gums usage: gums command [command-options] Commands: generateGrid3UserVoMap - Generate grid3-user-vo-map.txt for a given host. generateGridMapfile - Generate grid-mapfile for a given host. manualGroup-add - Includes a DN in a group. manualGroup-remove - Removes a DN from a group. manualMapping-add - Adds a DN-to-username in a mapping. manualMapping-remove - Removes a DN from a mapping. mapUser - Local credential used for a particular user. mapfileCache-refresh - Regenerates mapfiles in the cache. updateGroups - Contact VO servers and retrieve user lists. For help on any command: gums command --help`
- Now, you need to tell gums-admin where is your GUMS server.
 

```
[root@gums bin]# cat ../etc/gums-admin.properties
gums.location=https://localhost:8443/gums/services/GUMSAdmin Replace the localhost with the
full machine name (even if you installed GUMS Admin on the same machine). [root@gums bin]# vi
../etc/gums-admin.properties [root@gums bin]# cat ../etc/gums-admin.properties
gums.location=https://gums.mysite.com:8443/gums/services/GUMSAdmin
```
- You will run gums-admin as a normal user, and not as root. You will need to change permission on the directory:
 

```
[root@gums bin]# chown username:groupname .. -R
```
- Test the service by generating a mapfile at the command line
 

```
[root@gums bin]# su - username [username@gums bin]# grid-proxy-init [username@gums bin]#
./gums generateGridMapfile testing.test.gov
```

You should get the same response you got from the web server.

Congratulations you installed GUMS Admin. To make it actually useful, now you need to go back on the server and write an XML policy file. Please refer to the rest of the GUMS documentation.

## Install the host client

If you want to be able to generate the maps for a host on that host, you can install GUMS Host. Grab the latest gums-host-0.7dev-latest.noarch.rpm file from the [dist directory](#) : it will install by default in /opt/gums, but the package is relocatable, so you can install it wherever you want.

- Grab the latest build and install [root@gums root]# wget http://grid.racf.bnl.gov/GUMS/dist/gums-host-0.7dev-latest.noarch.rpm . [root@gums root]# rpm -Uvh gums-host-0.7dev-build<latest>.noarch.rpm
- Check that GUMS was installed correctly [root@gums root]# cd /opt/gums/bin/ [root@gums bin]# ./gums-host usage: gums-host command [command-options] Commands: generateGrid3UserVoMap - Generate grid3-user-vo-map.txt for this host. generateGridMapfile - Generate grid-mapfile for this host. mapUser - Maps a user to the local account using the AuthZ interface used by the gatekeeper callout. For help on any command: gums-host command --help
- Now, you need to tell gums-host where is your GUMS server. [root@gums bin]# cat ../etc/gums-admin.properties gums.location=https://localhost:8443/gums/services/GUMSAdmin gums.authz=https://localhost:8443/gums/services/GUMSAuthorizationServicePort Notice there are two addresses: one is the GUMS service and one is the GUMS Authorization service, which is the one also used by the gatekeeper callout. Replace the localhost with the full machine name [root@gums bin]# vi ../etc/gums-admin.properties [root@gums bin]# cat ../etc/gums-admin.properties gums.location=https://gums.mysite.com:8443/gums/services/GUMSAdmin gums.authz=https://gums.mysite.com:8443/gums/services/GUMSAuthorizationServicePort
- You will run gums-host as root, since it will authenticate with the host certificate. Test the service by generating a mapfile at the command line [root@gums bin]# ./gums-host generateGridMapfile  
The response will vary depending whether you have configured the policy or not.

Congratulations you installed GUMS host.

## Problems?

If you have any problem, feel free to contact GUMS developers.

## 1.3 Installation guide

---

### GUMS installation and walkthrough

*This article describes all the components of GUMS and all the installation details; it is not meant as a quick installation guide. **Please, refer to the quick installation guide if you prefer to get up and running and need the command line.** This guide assumes you are familiar with SQL, a bit of Java, Tomcat, ..., and that you are not interested in the exact command lines to write. It's meant for giving all the pieces of the puzzle so that an admin is comfortable with what GUMS does, and has an idea where to put his hands in case of problems.*

#### The pieces

Before beginning, we describe what components are there in GUMS.

- The service: the main component is a web application that consists of a Web Service interface (WS) and a Web User Interface (WebUI). The same web application contains web pages for the admin to use through a browser, and a Web Service door that allows command line tools and other services to use GUMS functionalities. The Web Service part is a SOAP service built on Apache Axis. The web application is secured through SSL, that is also both WS and WebUI have the same transport level security. If GUMS is installed on a server without SSL, though part of the WebUI will be available, it won't proceed with any operations. The authorization is internal to GUMS: in the configuration you define a group of admins, who have full access; Services will have only read access to their mappings (i.e. map user or generate maps).  
The service was developed and tested on a Tomcat 5.0.28 + EGEE security installation, though it should work on any J2EE compliant web servers with SSL.
- The persistence layer: GUMS will need a place to store some temporary data, for example it caches the information it reads from the VO server, and some critical data, such as manual mappings the admin might want to define. At this time, the whole persistence layer is based on a MySQL server, though we plan to allow any RDBMS (the persistence layer needs some refining before it can be used 24/7, as part of that refining we will be able to use almost any JDBC compliant database). The idea, though, is that a site might want to integrate this part within their infrastructure. For example, at BNL we want to store all the critical data on our LDAP servers, which already contain most of the user accounts information. To integrate one needs only to implement a couple of classes and change the configuration file, all of which can be done at runtime. The suggestion is to evaluate GUMS on a MySQL back-end, and then, if desired, plan the integration.
- The admin tools: this is a set of command line tools to administer GUMS. They connect to the WS to perform the different tasks. All operation will be carried with the admin credentials, who will use the GRID proxy beforehand. All this activity will be logged on the server. Being the admin tools are a client to GUMS, they can be installed on the same host where GUMS is running or on a different host.
- The host tools: this is a set of command line tools to be installed on a generic host to retrieve the maps (i.e. grid-mapfile et al) , or to test the connectivity for the callout. All operation must be carried

with the host/service credentials, that is the certificate and private key. The host will be allowed only to retrieve information about its mappings. The CN of the host certificate and the hostname will be used, and must match to the GUMS setup. That is, for a host gatekeeper.mysite.com, with a CN=gatekeeper.mysite.com, GUMS will have to have a map for the host with the same name.

GUMS can also run in different modes, described in the GUMS modes of operation section. They are essentially variations in the user of GUMS. We won't talk extensively about those, as they are getting less relevant now, as we get closer and closer to GUMS 1.0. We will mention a few things, though, that affect the setup. It will be indicated by a "*Different mode note:*" warning.

The installation steps are:

- Preparing the persistence layer backend (i.e. MySQL for the standard installation)
- Installing the service and setting up a policy
- Installing the admin tools
- Installing the host tools on the target gatekeepers

### Root vs non-root

GUMS can be installed as both root and non-root. The only issue is the host certificate: GUMS must be able to access a host/service certificate with its private key for authentication. Generally, it is located in /etc/grid-mapfile/hostcert.pem with root permissions. One could either set those permissions to a different user, or create another copy for gums. Such as /etc/grid-security/gumscert.pem.

### Firewall and security considerations

GUMS doesn't require any port to be installed outside the firewall. The only requirement is to have an inbound TCP port opened on the GUMS server (default 8443), and an outbound port from all gatekeeper to that GUMS port.

All GUMS requests are over SSL. Grid certificates are used for authentication and authorization.

All access to GUMS is logged. Logs can be configured to use syslogd, which can be used to forward the logs to the cybersecurity department of the site.

### Prepare the database

The first to do is to prepare the database. There is currently no script to do it: it's something we plan on doing.

You will need a mysql server, with version 4.0.18 or greater installed. You can either install one from scratch (follow the instruction on mysql's site) or you can use an installation you have ready.

Log in as root and run the following script:

```
CREATE DATABASE GUMS_0_7;  
  
GRANT ALL  
  ON GUMS_0_7.*  
  TO gums@'<gumserver.site.gov>' IDENTIFIED BY '<password>';
```

```

USE GUMS_0_7;

CREATE TABLE User (
  userID INTEGER AUTO_INCREMENT PRIMARY KEY,
  userDN VARCHAR(255) NOT NULL,
  userFQAN VARCHAR(255) DEFAULT NULL,
  registrationDate DATETIME NOT NULL,
  removalDate DATETIME DEFAULT NULL,
  userGroup VARCHAR(255) NOT NULL
);

CREATE TABLE UserAccountMapping (
  mappingID INTEGER NOT NULL,
  userDN VARCHAR(255) DEFAULT NULL,
  account VARCHAR(31) NOT NULL,
  startDate DATETIME DEFAULT NULL,
  endDate DATETIME DEFAULT NULL,
  userGroup VARCHAR(255) NOT NULL
);

```

in which you should replace <gumsserver.site.gov> with the machine on which you will run the GUMS server component. As you see the database is very simple: it's just a place to store lists of users (in the User table) and a list of mappings (UserAccountMapping). It's meant to be simple: it's direct consequence of the requirement that GUMS should be easily ported to other persistence mechanisms (i.e. LDAP, other site internal DB, ...). The first table will be mainly used to cache the values from the VO servers. For example, in the policy you will specify you want to map all the users from the ATLAS VO to the 'usatlas1' account: from time to time GUMS will query the VO server and store the list of users in the User table. For each mapping request, GUMS will look at its local copy instead of the remote VO server.

The User and UserAccountMapping tables are also critical for manual user groups and manual mappings. This means an admin is free to create a group of certificates, or a certificate to user mapping, to handle special cases. GUMS will have commands to add entries to these mapping, so you do not need to use the DB directly. You can if you want, though, for integration purposes. In any case, the use of these manual groups and mappings has a big effect: this is critical information that cannot be lost. Which means you will need to backup the server. This is a good candidate for integration: you might want to keep this information in the same information system you use for user account, as the information is connected.

To sum up: if you do not use manual groups and mappings, the information in the database can be regenerated at any time. If you do, GUMS has critical information, and you might want to make it safer through some kind of backup.

*Different modes note:* if you need to use mode 3 or 2 1/2 you will need to 2 extra table to store the maps. You should run:

```

CREATE TABLE HostGridMapfile (
  mapfileID INTEGER AUTO_INCREMENT PRIMARY KEY,
  hostname VARCHAR(255) NOT NULL,
  mapFile MEDIUMTEXT NOT NULL
);

CREATE TABLE HostGrid3UserVoMap (
  mapfileID INTEGER AUTO_INCREMENT PRIMARY KEY,
  hostname VARCHAR(255) NOT NULL,

```

```
mapFile MEDIUMTEXT NOT NULL
);
```

Once you created the database, you probably want to insert your DN in the User table within a group you will later use in GUMS as the admin group. For example:

```
INSERT INTO User SET userDN="/DC=org/DC=doegrids/OU=People/CN=Gabriele Carcassi",
userGroup="admins";
```

This is essentially in a manual user group named "admins".

The database is prepared: you can now proceed to install the Service

### Installing the service

GUMS is written in java, and requires java to be installed to run. It was developed against Sun JDK 1.4.2, but it should run on any 1.4.x and 1.5.x compliant JVM. If you need to install java, or learn more about it, refer to the documentation at <http://java.sun.com>.

The GUMS service is a standard J2EE application, which means it is a file (.war) that can be installed in any compliant engine. We provide a tarball containing a Tomcat 5.0.28 + EGEE security preconfigured, which is the configuration GUMS was developed against. The EGEE provides an SSL Socket Factory that Tomcat uses to create the SSL connections. The EGEE SSL is essentially standard SSL with the addition of Grid proxies. You can find more information about it if you look for "glite trustmanager". The bundled tomcat is modified in the following way:

- 4 more jars were put in the \$CATALINA\_HOME/server/lib directory:  
bcprov-jdk14-125.jar  
glite-security-trustmanager.jar  
glite-security-util-java.jar  
log4j-1.2.8.jar  
These contain the EGEE SSL socket factory and dependency
- In \$CATALINA\_HOME/conf the following file is added:  
log4j-trustmanager.properties  
Which is the logging configuration for the EGEE security
- The \$CATALINA\_HOME/conf/server.xml was modified. The following section is added:  

```
<Connector port="8443" maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
enableLookups="false" disableUploadTimeout="true" acceptCount="100" debug="0"
scheme="https" secure="true"
sslImplementation="org.glite.security.trustmanager.tomcat.TMSSLImplementation"
sslCAFiles="/etc/grid-security/certificates/*.0" crlFiles="/etc/grid-security/certificates/*.r0"
sslCertFile="/etc/grid-security/hostcert.pem" sslKey="/etc/grid-security/hostkey.pem"
log4jConfFile="/opt/gums-service/conf/log4j-trustmanager.properties" clientAuth="true"
sslProtocol="TLS" />
```

This makes Tomcat listen on port 8443 for https, using the EGEE security. You will notice all the parameters for a Grid connection, and the configuration file that was added for EGEE security



logging. The part declaring the port 8080 on http was closed.

Essentially, this is what is needed to setup EGEE security.

The web application is available on the web site as a war, though you should use it unpacked: the configuration files are for now kept in the /WEB-INF/classes directory, and loaded through classloading; using the war means you would need to modify the files within the war archive, which is not practical.

The configuration files are 2:

- log4j.properties. This is a standard log4j configuration, which will determine how the logging is implemented. This is loaded once when the server is started. To change it, the service will need to be restarted.
- gums.config. This is the policy file for GUMS, which determines how all users will be mapped to their local account. Also the access to the database is defined here. This can be changed at any point, and the service will pick it up. At any operation that needs the configuration, a check is performed to see if the file has been changed. If it is, this will trigger a configuration reload.

*Different modes note:* hosts.conf: this is used in modes 2 1/2 and 3 to determine which maps should be created and stored in the mapfile cache

You can refer to the full documentation of the configuration files for the details.

Once you have setup the server and the web application, you can use your browser with your grid credential to connect to the WebUI and run a couple of commands.

### Installing the admin tools

The admin tools come packaged in an rpm. The default location is /opt/gums, but the destination can be changed. Once you install it, you will see 4 directories:

- bin: contains the gums executable. It's a shell script that prepares the java environment for GUMS. One critical part is the creation of the variable to handle the grid security. It is a single executable that accept different commands, like cvs does (i.e. `./gums mapUser ...`, `./gums generateGridMapfile ...`). The script has `--help` options that explains what are the current features.
- etc: contains the configuration file for admin, which is one. The only thing it contains the URL to the servlet that runs the web service. You have to point to your server before being able to run any commands
- lib: contains all the java libraries needed by GUMS
- var/log: contains the log files for the admin. There won't be much there, as all the functionalities are implemented on the service.

You can use the admin to:

- View the generations of the map
- Change the manual group and mapping
- Trigger a refresh of the groups (i.e. make GUMS contact the VO servers to refresh the local member lists)

### Installing the host tools

The host tools also come package in an rpm. The default location is also /opt/gums, and the destination can be changed. The directory structure is the same than the admin tools. The differences are:

- The command is gums-host
- The script will set the credentials as the one for the host (i.e. /etc/grid-security/hostcert.pem, ...)
- The set of commands are limited

You can use the host tools to:

- Generate maps for the host: these will be based on the hostname
- Test the connectivity of the callout door. TODO This still needs to be implemented.

If you cannot use the callout on a gatekeeper, you will be able to use the host tools within a cron job to update the maps at regular interval.

## 1.4 Configuration

---

### Configuring GUMS

There are three files used by GUMS for the configuration.

- gums.config - holds the policy used by GUMS to create the mapping. It defines where to get the VO members, how should they be mapped, and which groups can access which gatekeepers
- hosts.conf - contains a list of hosts for which to generate the mapfile for the cache (not used if using GUMS service)
- db.properties - contains the db information for gums host direct access (not used if using GUMS service)

## 1.4.1 gums.config

---

### **gums.config**

This file contains the policy in an XML format. The syntax is meant to allow anybody to create his/her own components and integrate them just by dropping a jar in the lib directory. Therefore many components are defined by class names and bean properties. (If you are not a java programmer, a bean property is a getXxx/setXxx pattern, where xxx is the name of the property).

The XML file has this structure:

```
<gums>
  <persistenceFactories>
    <persistenceFactory/>
  </persistenceFactories>
  <adminUserGroup/>
  <groupMappings>
    <groupMapping>
      <userGroup/>
      <accountMapping/>
    </groupMapping>
    <groupMapping>
      <userGroup/>
      <compositeAccountMapping>
        <accountMapping/>
        <accountMapping/>
        <accountMapping/>
      </compositeAccountMapping>
    </groupMapping>
    ...
  </groupMappings>

  <hostGroups>
    <hostGroup/>
  </hostGroups>
</gums>
```

There are 3 sections:

- persistenceFactories - defines where the local data can be stored. For example, GUMS will keep a local copy of the VO listings: you can decide where to keep them. Each component that will need persistence will retrieve it through the factory. This allows to create a custom persistence layer for the facility.
- groupMappings - defines group of user and how they are mapped. A groupMapping is defined by two thins: a set of users (userGroup) and a policy for account mapping (accountMapping). Optionally, the policy can be composed by different policies (compositeAccountMapping)
- hostGroups - defines which groupMappings are used for the different hosts

## persistenceFactories

This section just contains a list of persistenceFactory elements.

```
<persistenceFactories>
  <persistenceFactory name='mysql'
    className='gov.bnl.gums.MySQLPersistenceFactory' />
</persistenceFactories>
```

## persistenceFactory

The type of persistenceFactory is determined by the class which has to implement the PersistenceFactory interface. The basic attributes are:

Attribute	Description	Examples
name	The name that will be used by the other components to refer to this persistenceFactory.	mysql files ldap
className	The class that is going to provide the implementation for the persistence layer. It must implement gov.bnl.gums.PersistenceFactory.	gov.bnl.gums.MySQLPersistenceFactory org.mysite.HRDatabaseFactory

Other attributes are implementation specific.

All the elements that will be using the database, will need to set the 'persistenceFactory' attribute to the name, and then provide a 'name' attribute that will identify which information to use. What that name means is implementation specific. For a database layer, for example, it could mean a table or a column value within a well known table; for a file layer it could mean the name of the file.

## gov.bnl.gums.MySQLPersistence Factory

Currently, the only implementation provided is the MySQLPersistenceFactory. All the attributes are passed as properties to the database driver. For example:

```
<persistenceFactory name='mysql' className='gov.bnl.gums.MySQLPersistenceFactory'
  jdbcDriver='com.mysql.jdbc.Driver'
  jdbcUrl='jdbc:mysql://mydb.mysite.com/GUMS_0_7' user='gums'
  password='mypass' autoReconnect='true' />
```

## adminUserGroup

This defines the set of users that have admin privileges on GUMS. This entry has the same options as a userGroup entry. Refer to that part of the documentation.

## groupMappings

This section will contain a list of groupMappings elements.

### groupMapping

A group mapping is composed by two elements: a userGroup and a mapping, which can either be a compositeAccountMapping or an accountMapping.

Attribute	Description	Examples
name	The name that will be used by the other components to refer to this persistenceFactory.	atlas star phenix

### userGroup

This element defines the list of people that will be part of this groupMapping. A userGroup is typically defined by a group on a VO server or on a database. This element corresponds to the UserGroup interface in the code, meaning you can provide your own logic. The basic attributes are:

Attribute	Description	Examples
className	The class that is going to provide the implementation for the user group. It must implement gov.bnl.gums.UserGroup.	gov.bnl.gums.LDAPGroup gov.bnl.gums.VOMSGroup gov.bnl.gums.ManualGroup

### gov.bnl.gums.LDAPGroup

This class retrieves the list of members from an LDAP VO, as it is defined within LCG. The attributes available are:

Attribute	Description	Examples
server	The LDAP server from which to retrieve the information	grid-vo.nikhef.nl
query	The query to be used on the server.	ou=usatlas,o=atlas,dc=eu-datagrid,dc=org ou=People,o=atlas,dc=eu-datagrid,dc=org
persistence Factory	The persistence layer to be used to store locally the list of users. The string must be one of the names defined within the persistenceFactories section.  GUMS doesn't contact the server at every request, but it keeps a local cache, which is refreshed from time to time	mysql

Attribute	Description	Examples
name	The name of the cache within the persistence factory. Refer to the specifics of the persistence factory itself.	atlas usatlas

For example:

```
<userGroup className='gov.bnl.gums.LDAPGroup' server='grid-vo.nikhef.nl'
query='ou=People,o=atlas,dc=eu-datagrid,dc=org' persistenceFactory='mysql'
name='atlas' />
```

Retrieves all the user in the ATLAS VO LDAP server.

### gov.bnl.gums.VOMSGroup

This class retrieves the list of members from an VOMS Server. The attributes available are:

Attribute	Description	Examples
url	The url of the web services for VOMS. Notice that it needs the full url of the service: it won't be constructed from the server name or vo.	https://vo.racf.bnl.gov:8443/edg-voms-admin/atlas/services/VOMSAdmin
voGroup	The group defined within the VO.	/atlas/test /atlas/group/subgroup
persistence Factory	The persistence layer to be used to store locally the list of users. The string must be one of the names defined within the persistenceFactories section.  GUMS doesn't contact the server at every request, but it keeps a local cache, which is refreshed from time to time	mysql
name	The name of the cache within the persistence factory. Refer to the specifics of the persistence factory itself.	atlasTest atlasGroupSubgroup

For example:

```
<userGroup className='gov.bnl.gums.VOMSGroup'
url='https://vo.racf.bnl.gov:8443/edg-voms-admin/atlas/services/VOMSAdmin'
persistenceFactory='mysql'
name='atlas' voGroup="/atlas/test"
sslCertfile='/etc/grid-security/hostcert.pem'
sslKey='/etc/grid-security/hostkey.pem' />
```

Retrieves all the user in the VOMS server at the specified url from the /atlas/test group. It also specifies which credentials should be used to contact the server.

### gov.bnl.gums.ManualGroup

This class manages a group of identities stored in the persistence. Useful to handle special cases, for development testbed or for the list of admins. GUMS The attributes available are:

Attribute	Description	Examples
persistence Factory	The persistence layer to be used to store the list of users. The string must be one of the names defined within the persistenceFactories section.	mysql
name	The name of the group within the persistence factory. Refer to the specifics of the persistence factory itself.	test testbedA admins

For example:

```
<userGroup className='gov.bnl.gums.ManualUserGroup' persistenceFactory='mysql'
name='testGroup' />
```

Selects the users stored manually in the testGroup group.

### compositeAccountMapping

A compositeAccountMapping is a mapping policy made up by a list of policies. When a request to map a user comes, the composite mapper will forward the request to the first mapper in the list. If this fails, the request is forwarded to the second, and so on. This allows you to create a policy that has a default (the last element on the list) but allows special cases (the top element in the list).

This element simply contains a list of accountMapping elements.

### accountMapping

An account mapping defines the logic with which the user credentials are mapped to the local account. The logic will be provided by a class implementing the gov.bnl.gums.AccountMapping interface.

Attribute	Description	Examples
className	The class that is going to provide the implementation for the mapping. It must implement gov.bnl.gums.AccountMapping.	gov.bnl.gums.ManualAccountMapper gov.bnl.gums.NISAccountMapper gov.bnl.gums.AccountPoolMapper gov.bnl.gums.GroupAccountMapper

### gov.bnl.gums.NISAccountMapper

This class retrieves the NIS maps and tries to match the name from a certificate. Please, read the full documentation on the javadoc about this class before using it. The attributes available are:



Attribute	Description	Examples
jndiNisUrl	The url as defined in the Java JNDI driver, that allows to specify the NIS server and the domain.	nis://nis.bnl.gov/atlas

For example:

```
<accountMapping className='gov.bnl.gums.NISAccountMapper2'
jndiNisUrl='nis://nis.mysite.org/domain' />
```

Uses the NIS map taken from the nis.mysite.org server for domain.

### gov.bnl.gums.AccountPoolMapper

This class implements account pooling. Please refer to the account pool documentation for the full detailed description. The attributes available are:

Attribute	Description	Examples
TODO		

For example:

```
TODO
```

todo

### gov.bnl.gums.GroupAccountMapper

This class maps all users to the same account. The attributes available are:

Attribute	Description	Examples
groupName	The name of the account	atlas testAccount

For example:

```
<accountMapping className='gov.bnl.gums.GroupAccountMapper' groupName='atlas' />
```

Maps everyone to the atlas account.

### hostGroups

This section contains a list of host groups. To determine to which group a particular host is part, GUMS start from the first one in the list and stops at the first match.

### hostGroup

A hostGroup defines a group of hosts and which groupMappings will be used. This element corresponds to the HostGroup interface. This allows to retrieve hosts lists from other components of the facility, for example an information service.

Attribute	Description	Examples
className	The class that is going to provide the implementation for the hostGroup. It must implement gov.bnl.gums.HostGroup.	gov.bnl.gums.WildcardHostGroup
groups	A list of groupMappings, in the order of preference. To determine which group should be used for a particular user, GUMS will start from the beginning of the list until it finds a match. Therefore, if there would be more than one match (i.e. a user is part of more groups) the first one in the list is used.	group1,group2

### gov.bnl.gums.WildcardHostGroup

This class represent a set of hosts defined by a hostname wildcard. For example, \*.mysite.org would include all the hosts which end in mysite.org. The attributes that can be set for this class are:

Attribute	Description	Examples
wildcard	The wildcard for the set of hosts to be included. The wildcard is a string where * can be substituted with any character, except '.'. That is, *.bnl.gov wouldn't match myhost.usatlas.bnl.gov.	myhost.mysite.org atlas*.mysite.org *.atlas.mysite.org

For example:

```
<hostGroup className="gov.bnl.gums.WildcardHostGroup"
wildcard='*.usatlas.bnl.gov' groups='gridex,sdss,uscms,usatlasGroup,btev,ligo,ivdgl'
/>
```

Maps the hosts in the usatlas subdomain at BNL to the list of groups, which will have been defined in the groupMappings section.

## 1.4.2 db.properties

---

### db.properties

This file contains the configuration for the connection to the database. It is the same syntax for both the server and the client. It is basically the property file that will be sent to the JDBC driver plus two other properties that allow to select the driver. The database syntax was only tested on MySQL, and this is the only database supported out of the box.

This is scheme temporary: once we make a transition to a grid service, there will be no need for the client to use the database. Also, we plan to reimplement the persistence layer using EJB to allow clustering. In that case the datasource will be configured through JNDI as any other J2EE application, and the database supported will be all the ones supported by the application server used.

```
jdbc.driver = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://hostname/GUMS
user = gums
password = mypass
```

Those accustomed to JDBC will recognize the standard jdbc syntax for the driver and url. More information about the mysql driver connection parameters should be found at <http://dev.mysql.com/doc/connector/j/en/index.html>

## 1.4.3 hosts.conf

---

### Using hosts.conf

*The hosts.conf files is used only in mode N 3 and N 2 1/2. **Most people can completely disregard this file.** We are just being complete in the documentation.*

This file contains the list of hosts for which gums is going to create the mapfiles in the cache. As of now, the server components stores the maps in the database when the gums-mapfilecache-refresh is executed. This command relies on hosts.conf to know for which hosts to create the maps. The client will use the host of the machine on which is installed when running gums-mapfilecache-retrieve. If the client has multiple names due to multiple interfaces, you should use the name you see once logged in the hosts (i.e. run 'hostname').

The configuration file is simply a text file with the full name of each host on each line. For example:

```
grid01.mysite.org
grid02.mysite.org
testgrid.mysite.org
```

## 1.5 Commands: GUMS Admin

---

### GUMS Admin commands

GUMS admin consists of a set of command line tools which will be run under the user GRID credentials. The user must be part of the GUMS admins.

#### gums

The command is gums with its many options. To see all the options available:

```
[root@gums bin]$ ./gums
usage: gums command [command-options]
Commands:
generateGrid3UserVoMap - Generate grid3-user-vo-map.txt for a given host.
generateGridMapfile - Generate grid-mapfile for a given host.
manualGroup-add - Includes a DN in a group.
manualGroup-remove - Removes a DN from a group.
manualMapping-add - Adds a DN-to-username in a mapping.
manualMapping-remove - Removes a DN from a mapping.
mapUser - Local credential used for a particular user.
mapfileCache-refresh - Reegerates mapfiles in the cache.
pool-addRange - Adds accounts to an account pool.
updateGroups - Contact VO servers and retrieve user lists.
For help on any command:
gums command --help
```

### Service mapping generation commands

This set of commands can be used by the admin to check how the mapping across the services is maintained. One can look how the maps generated by GUMS look like, and check to which local user any Grid identity is mapped.

#### gums mapUser

With this command an admin can check the mapping of a specific identity, including the VOMS extended proxy FQAN. This allows to check the user is mapped to the correct account when using different VO roles.

```
[maven@gums bin]$ ./gums mapUser --help
usage: gums mapUser [-h HOSTNAME] [-f FQAN] USERDN1 [USERDN2] ...
Maps the grid identity to the local user.
Options:
  -f,--fqan <arg>      Fully Qualified Attribute Name, as it would be
                        selected using voms-proxy-init; no extended information by
```

```
default
-h,--hostname <arg>  hostname; localhost by default
--help               print this message
```

### **gums generateGridMapfile**

This shows the grid-mapfile GUMS generate for a given host.

```
[root@gums bin]$ ./gums generateGridMapfile --help
usage: gums generateGridMapfile [-f FILENAME] [HOSTNAME]
Generates the grid-mapfile for a host. HOSTNAME is the name of localhost
by default.
Options:
-f,--file <arg>      saves in the specified file; prints to the console by
                      default
--help               print this message
```

### **gums generateGrid3UserVoMap**

This shows the generated inverse vo map used by grid3 for accounting. One can check host per host the product of the generation.

```
[root@gums bin]$ ./gums generateGrid3UserVoMap --help
usage: gums generateGrid3UserVoMap [-f FILENAME] [HOSTNAME]
Generates the grid3-user-vo-map.txt for a host. HOSTNAME is the name of
localhost by default.
Options:
-f,--file <arg>      saves in the specified file; prints to the console by
                      default
--help               print this message
```

## 1.6 Logging

---

### Log

GUMS is designed with 3 logs in mind: developer's log, administrator's log, site security log. This means that you won't find the same things in all of them, and you shouldn't. For example, say that GUMS connects to a VO server to retrieve a list of users, and the VO server replies with an empty list. From the developer's perspective the code has worked fine; but from an administrator's perspective it's probably the sign that something not going well.

The logs come with a predefined configuration, which is what we describe here. To know more of the details, especially how to change the configuration, refer to the logging implementation.

#### Administrator's log

This log is meant for the maintainer of GUMS at a particular site. He is responsible of installing and configuring GUMS. To manage the mapping by keeping all the information up-to-date.

The log is placed under the service directory (`/opt/gums-service/logs/gums-resource-admin.log`). The log can also be configured to be forwarded by mail in case of error. This is particularly useful as the admins can be informed right away of any problem. Look at the `log4j.properties` configuration file.

The log includes every command that is being executed by any admins. This allow the administrator to keep full control of what is happening, together with a history of what has happened to be able to troubleshoot automatic procedures. The main features are:

- All successful commands are logged as INFO with both input and output parameters
- All unsuccessful commands (including failure do to authorization) are logged as ERROR

#### Developer's log

The developer log is meant for someone developing the code or fixing bugs.

The log is located under the service directory (`/opt/gums-service/logs/gums-developer.log`).

#### Site security log

The site security log is meant for the cybersecurity department of a lab. It includes [TODO check requirement and implement] all the information for auditing the GUMS service. This information will be limited to accesses to the service that are going to modify the state of the service. All the access to the information will be typically already be logged at the gatekeeper.

The log can be configured to be forwarded to the AUTHPRIV facility of syslogd. To enable logging to the syslogd daemon, you have to modify the `log4j.properties` and make sure it allows logging from the

network. To enable logging from the network, you need to start syslogd with -r option. [root@atlasgrid13 log]# cat /etc/sysconfig/syslog # Options to syslogd # -m 0 disables 'MARK' messages. # -r enables logging from remote machines # -x disables DNS lookups on messages recieved with -r # See syslogd(8) for more details SYSLOGD\_OPTIONS="-r -m 0" # Options to klogd # -2 prints all kernel oops messages twice; once for klogd to decode, and # once for processing with 'ksymoops' # -x disables all klogd processing of oops messages entirely # See klogd(8) for more details KLOGD\_OPTIONS="-x"

The reason is that Apache log4j SyslogAdapter can only log through the network (to allow portability), even if you are logging to the localhost.

Another possibility is to log directly to a remote server: you can do that by modifying the log4j.properties configuration file in the service.



## 1.6.1 Logging implementation

---

### Log implementation

All information in GUMS is logged through the apache commons logging package. The implementation used in GUMS is apache log4j. To change the logging implementation you have to refer to the commons.logging implementation. Be aware that some library that GUMS uses may not be as well behaved in regard to logging.

The configuration is controlled by the log4j.properties file. This is a normal log4j configuration file: refer to the log4j manual for more information.

GUMS using the follow conventions for log names:

- The developer's log uses one log for each different class, with the name being the class name. Given GUMS package structure, "gov.bnl.gums" contains the whole development log for GUMS. This allows the develop to filter the log of the code he is working on.
- The admin log uses the log named "gums.resourceAdmin"
- The site security log is at "gums.siteAdmin"

### Administrator's log

This log is meant for whoever is maintaining GUMS installation at a particular site. The log is designed to be used in different ways: on standard error, in a log file and in e-mails. E-mails will get from WARN up, the standard error will receive from INFO up and the log can go down to TRACE. The breakdown on the logging level is:

TODO: The admin log still needs a little thinking

- FATAL - GUMS is unable to operate: no functionalities are available. For example, a configuration file was not written correctly.
- ERROR - A particular operation failed or was incomplete. For example, the CMS VOMS server couldn't be contacted, so it's members weren't refreshed (even though the ATLAS group was); the NIS server didn't respond, so it is impossible to generate the grid-mapfile for atlasgrid25, though the grid-mapfile for atlasgrid26 could be generated since it doesn't require the NIS information.
- WARN - A condition that hints to a misconfiguration or incorrect usage. For example, a VO server returned no users.
- INFO - A condition that might hint to a problem, but is not critical per se. For example, the NIS mapper couldn't find a match. A big difference is that INFO doesn't trigger a mail, while WARN does. It is preferable to log many similar problems as info and then send a WARN to actually send the mail.
- DEBUG - Not used

- TRACE - Not used

### Developer's log

The developer log is meant for someone developing the code or fixing bugs. Each class will use the log named as their full class name. The breakdown on the logging level is:

- FATAL - An exception or an inconsistency that forces GUMS to terminate or not function at all. For example, a configuration file was not written correctly or couldn't be found.
- ERROR - An exception or an inconsistency that doesn't allow GUMS to complete a particular operation or part of it. For example, the CMS VOMS server couldn't be contacted, so it's members weren't refreshed (even though the ATLAS group was); the NIS server didn't respond, so it is impossible to generate the grid-mapfile for atlasgrid25, though the grid-mapfile for atlasgrid26 could be generated since it doesn't require the NIS information.
- WARN - An exception or an inconsistency that is not necessarily going to affect functionalities, or an error condition that was recovered. For example, a particular cache was found to be out of synch and was rebuilt.
- INFO - The successful completion of a macro-event (i.e. something that happens only once in a while). For example, the configuration file was read, the server was started. Typically used to debug configuration problems.
- DEBUG - The attempt or successful completion of a smaller event. For example, a query was executed, a user was mapped. Typically one shouldn't have more than one DEBUG statement in a method.
- TRACE - Shows the internal execution of the code. As a contrast, building a query would be at this level. Inside method logging should be done at this level

### Site security log

The site security log will log all accesses.

- INFO - Will log all the "write" accesses
- DEBUG - Will log all the "read" accesses

## 1.7 Integration

---

### GUMS extension and site integration

*All the main components of GUMS are developed against interfaces with minimal coupling to GUMS itself, allowing a site to rewrite those components to interface their systems. In this article we will describe those interfaces and provide examples on how this integration can be done. We refer to the online GUMS code for examples: **if you want to read this on printed paper, you might also want to print the code linked from the online version of this article.***

The policy file, as you might know by now, allows to select at runtime the different pieces of code that performs crucial functionality during the mapping. This allows a site admin to extend GUMS to talk to its site information systems. For example, at a particular site an admin might require to:

- Store all the information GUMS uses for the mapping in the same system used by the rest of site account (i.e. LDAP, databases like Oracle or MySQL, ...)
- Use an already existing software, developed within the site, to perform the mapping (i.e. as part of the user information database, a user was already able to select the grid certificate at the site)
- Use a different database for group information (i.e. the site wants to map his admins in a different way, and wants to take the list of admins directly from its databases)
- Use some other information service to decide which service should use which mapping (i.e. one wants to set on all production machines a particular mapping, and the list of production machines is stored within their own information service)

GUMS doesn't require integration: it can work fine by itself. But if a site requires integration, it is easy to do: it just requires a little knowledge of Java. If your use case is not site specific, we are willing to help either in the development or to distribute it as part of GUMS. Go to the GUMS site and use our mailing lists!

### Changing storage for GUMS data

Suppose you have already a user management system that already keeps some grid to username mapping, or that you want to tightly couple the pool account system that GUMS provide with your site LDAP, or that you want to give a special mapping to a list of users which can be taken directly from some database: how would you do it, you might ask.

In GUMS, there is a `PersistenceFactory` class which is responsible to create all the objects that are going to write to/from a specific information system. For example, GUMS provides a `MySQLPersistenceFactory` which is going to implement the logic of how to write/read on the default MySQL implementation. A site can implement its own `PersistenceFactory` to rewrite where the data is written and read from.

The `PersistenceFactory` is just a factory class to create the objects that actually implement the persistence layer. There are different kind of those objects: `UserGroupDB` is used by the many groups to cache lists of users so GUMS doesn't have to contact the VO server every time; `ManualAccountMapperDB` is used

by the `ManualAccountMapper` to store a mapping table. The first note is that you do not need to implement all the kinds, but only the one or more you need. For those you do not need to implement, you can use:

```
throw new java.lang.UnsupportedOperationException(...);
```

This will allow you to trap and recognize mistakes in configuration later. This is also true for any other method you do not feel to implement (i.e. you might not want GUMS to modify the information in your site databases)

You can use the `MySQLPersistenceFactory` as a trace: it creates an inner class for any interface it needs to implement, and when asked to give a persistence object it just creates an instance of the appropriate inner class. You can, this way, implement a `ManualAccountMapper2DB` that is going to read the site user management system. You can implement an `AccountPoolMapperDB` that reads the account informations from the site LDAP. You can implement a `ManualUserGroupDB` that reads the list of users from your database.

### Creating a mapping

One of the things a site can do, is to create its own policy for the mapping. We suggest to look at the code of the mappings provided by GUMS as examples.

All the mappings implement the `AccountMapper` interface. The only method one needs to implement is the `mapUser`, which given the user credential is going to return the username.

The `GroupAccountMapper`, for example, will map all the users to the same account. The actual account will be set through the `groupName` property (notice the `getGroupName` and `setGroupName`) in the configuration file. For example:

```
<accountMapping className='gov.bnl.gums.GroupAccountMapper' groupName='test' />
```

will set the `groupName` to "test". You can create any property you like: while reading the configuration GUMS will look at your class for a name match [This is actually provided by the Apache Jakarta Commons Digester library].

We suggest you develop and test the class by itself, without running it in gums. You can have a main method, or a set of unit tests, to simulate some requests using the `mapUser` method, and see that it behaves correctly. Once you have done that, you can prepare a jar, and put it in the lib directory of the gums service. You will have to restart the service, as tomcat creates the list of available jars when the service is started. You can change the policy configuration, instead, at any time.

A mapping is particularly useful for those sites that have already a user management system, which already contains a username/certificate mapping. Through an extension, GUMS can be made to use that mapping.

### Creating a group

Another thing one can do is create their own group. Suppose, for example, that a site wants to use a new VO server, which is not, at that time, supported by GUMS. Or suppose that the site wants to grant all its admins a different mapping, and wants to get this list directly from their LDAP system, so that there is only one place to keep updated. All of these can be supported by creating a group.

Look at the `UserGroup` class: a group is essentially a function that is able to tell: "is this person in the group?". The `isInGroup()` function is going to check, by whatever means, if the Grid Identity is within that group.

There is a `getMemberList()`. This is actually optional: if you have an `EveryoneGroup`, for example, you can't name everyone. In that case, you should use something like:

```
throw new java.lang.UnsupportedOperationException("Group cannot be enumerated.");
```

The catch is that GUMS won't be able to generate a grid-mapfile for those hosts that make use of this group. Only the gatekeeper callout can be used.

The `updateMembers()` function is intended for those group that require access to remote services. For example, access to a VOMS server cannot be on a per request basis. For similar cases, one should implement the `updateMembers()` function to retrieve and store the data on a site local service. If you do this, you might want to use a `UserGroupDB` to store the information, so that it integrates with the rest of persistence. You can look at the `VOMSGroup` class for an example.

## 1.8 Operation modes

---

### GUMS modes of operation

*Here we describe the 3 ways GUMS was designed to operate. The second and the third were designed as both a gradual change and a fall-back solution. Here we provide a full explanation in case that someone is forced (or prefers) to use one of those modes.*

This article doesn't attempt to give a full explanation: we currently do not envision any new user starting from anything than the first mode. All these configuration have been part of an evolving system, which might be completely disregarded by the new user. We leave here for full documentation, and for fall-back solutions in case a site has to maintain a hybrid system. The advantage of being able to switch from one configuration to the other is that the policy won't change, mainly just software configuration on the gatekeepers.

#### 1. GUMS Service and GT2.x callout

In this configuration you have a GUMS Service, running all the time, that waits for the gatekeeper to call it. Using the callout, the gatekeeper will contact the GUMS service at every request to retrieve the local account. This is the desired mode of operations, as this allows most features, including Role based authentication. Also, all policies that cannot provide the full enumeration of all authorized users (i.e. all certificates from the DOEGrids CA, or default allow to a special account) cannot be implemented with the grid-mapfile.

To setup this mode, one sets up a normal GUMS Service, and configures the callout module provided in VDT to contact their GUMS Service.

#### 2. GUMS Service and grid-mapfile

In this configuration you have a GUMS Service, running all the time, that waits for a script running on the gatekeeper to contact it. GUMS Host will run as a cron job on the gatekeeper, and will retrieve a new version of the grid-mapfile.

To setup this mode, one sets up a normal GUMS Service, and installs GUMS Host on all gatekeepers. Then one would setup a cron job to run the "gums-host generateGridMapfile" command and redirect it to the grid-mapfile location.

The only difference in this mode is the client. In fact, one could have the same GUMS Service serving two clients with different modes. It can be used to make a more gradual transition.

#### 3. GUMS Admin and grid-mapfile cache

In this configuration there is no GUMS Service. The server part is done by a database server. GUMS

Admin is run directly, and pre-generates the maps for all hosts and stores them in the database. GUMS Host runs on the client, and retrieves the mapfiles from the database.

To setup this mode, one installs GUMS Admin on one machine, and GUMS Host on all gatekeepers. The gums-admin.properties should be set to direct: this will make GUMS admin run the command implementation directly on the machine, without contacting a service to run the command. For this purpose, GUMS admin will need the gums.config and hosts.config configuration files properly set in GUMS/etc. On GUMS Host, one set gums-admin.properties to direct also, and prepares a db.properties file with the database configuration. GUMS Host would contact the database to retrieve the map, instead of contacting the GUMS Service. Notice that the db access should be different from GUMS Admin and GUMS Host to avoid spreading critical information.

At this point, this mode is discouraged: it was the first one to be developed, since historically GUMS was created to managed the maps of a whole site. It is discouraged because it needs server-side changes to change mode. There is a better intermediate solution, which is the following.

## **2 1/2 GUMS Service and grid-mapfile cache**

This is an hybrid between 2 and 3, and it's intended as a passage from a system that has 3 implemented and wants to go to 2, or viceversa. In this mode, there is a GUMS Service that updates the database containing the maps for the whole site. A gatekeeper can retrieve the map from the database instead of from the GUMS Service directly.

To setup this mode, one sets up a normal GUMS Service, and runs "gums mapfile-refresh" within a cron job. The gatekeeper setup is as in 3.

## 1.9 FAQ

---

### Frequently Asked Questions

#### General

1. [What is GUMS?](#)
2. [Is GUMS being used in production anywhere?](#)
3. [I hear GUMS allows you to have different mappings on different gatekeeper. Why do you want to do that? Doesn't it complicate things?](#)

#### Using GUMS

1. [Does GUMS have to run at root?](#)

#### Building GUMS

#### Comparing GUMS with other tools

1. [What's the difference between GUMS and VOMS \(or VOMRS\)?](#)
2. [What's the difference between GUMS and using grid-mapfiles?](#)
3. [What's the difference between GUMS and edg-mkgridmap?](#)
4. [What's the difference between GUMS and LCMAPS?](#)

### General

#### General

What is GUMS?

GUMS is a Grid Identity Mapping Service that manages how Grid identities (i.e. certificates) are mapped to local identities. It's main two functionalities are creating grid-mapfiles and mapping a single user for a specific server. It's intended for a site that have multiple gatekeepers and wants to enforce a policy defining who has access to those resources and with what account. You can say things like: "on all the nodes that match \*.exp1.site.org, the users defined in group 'admin' in the 'ATLAS' VO are going to be mapped to a group account 'adminAtl1', while all the other users defined in the 'ATLAS' VO are to be mapped to their local account".

Is GUMS being used in production anywhere?

Yes, GUMS is being used at BNL for RHIC and ATLAS gatekeepers.

I hear GUMS allows you to have different mappings on different gatekeeper. Why do you want to do that? Doesn't it complicate things?

It's not that I want to... it's that I have to. At BNL, first, we have different gatekeepers for different experiment which have different needs. Plus, each experiment might have some gatekeepers in



productions, and some other in testing, and require slightly different configuration. Then comes troubleshooting: the time in which a user claims he cannot do a certain operation, and you might want to temporarily change your (the admin) mapping to his. Then comes the time in which you want to experiment with a new policy. All of these could be solved by using attributes within the VO servers, or by other tricks, which have the tendency of not being here right now. GUMS actually helps in keeping the mapping identical at all gatekeepers: this is what we generally want to do at BNL and one of the reason we wanted GUMS. But we still need to cope with the "irregularities" that a production environment has.

## Using GUMS

### Using GUMS

Does GUMS have to run at root?

No.

It will run as any user you will be running the tomcat server as.

## Building GUMS

### Building GUMS

## Comparing GUMS with other tools

### Comparing GUMS with other tools

What's the difference between GUMS and VOMS (or VOMRS)?

First of all, GUMS is a **site** tool while VOMS is a **VO** tool: you have a BNL GUMS, and you can have an ATLAS VOMS. A VO uses VOMS to keep a list of members, and their role within the organization. A site uses GUMS to maintain the mapping between GRID credentials (certificates) and local site credentials (i.e. UNIX accounts).

GUMS can contact VOMS to retrieve the list of VO users that needs a certain mapping. For example, you can say: all ATLAS members should be mapped to the "atlas" account. GUMS would contact the ATLAS VOMS server to know who are all these ATLAS members.

What's the difference between GUMS and using grid-mapfiles?

Using grid-mapfiles by itself is typically good only in testing environments. Usually one generates them with an external tool depending on the information present in the VO servers. For example, an external tool would contact the ATLAS VO, download the list of current users, and add them to the grid-mapfile.

GUMS can be used to generated grid-mapfiles, though it can also be used to replace grid-mapfiles. In that configuration, the gatekeeper contacts GUMS directly when it needs a mapping, instead of looking to the file

What's the difference between GUMS and edg-mkgridmap?

Edg-mkgridmap and GUMS host tool both create a grid-mapfile for a host. The first connects directly to the VO servers to retrieve the members and generates the map. The second is going to contact the GUMS server to retrieve an already prepared grid-mapfile.

GUMS gives a way to centrally manage grid access and the mapfile generation. The policy you can write in GUMS is also richer than the one in edg-mkgridmap. You can also use GUMS for both grid-mapfile generation and in conjunction with a gatekeeper (or service) callout. For a small site with a simple configuration, edg-mkgridmap might be a simpler solution. For a bigger site, with a more complicated environment, GUMS will give you more control and flexibility.

What's the difference between GUMS and LCMAPS?

WARNING: I am not an expert in LCMAPS. This is my understanding of the differences.

The short (and not 100% precise) answer is: GUMS is a Policy Decision Point while LCMAPS is a Policy Enforcement Point. The longer answer is: GUMS allows you to set a policy at the site level for all your gatekeepers or resources. It's a service that sits there and receives questions like: "Who should I map this guy to?". It doesn't actually enforce the mapping. In fact, GUMS by itself is useless if there isn't anything else that contacts to either retrieve a grid-mapfile or to request a specific mapping. The GUMS server is associated with clients that do that (i.e. the GUMS host tools, or the gatekeeper callout).

LCMAPS, instead, is inside the gatekeeper (or the gridftp server), it implements the callout, it decides and enforces the mapping. But there is one of them for every gatekeeper. There is no central mapping, no central policy. You configure each gatekeeper individually.

They are two different things, even though they both implement some similar functionalities. And, also, there is no reason why LCMAPS can't be the PDP for GUMS (i.e. LCMAPS could connect to GUMS as part of its decision process).

## 1.10 Changes








---

### Release History









Version	Date	Description
<a href="#">0.7.1</a>	in CVS HEAD	
<a href="#">0.7</a>	2005-01-14	
<a href="#">0.6.1</a>	2004-08-10	
<a href="#">0.6</a>	2004-07-01	
<a href="#">0.5</a>	2004-05-20	
<a href="#">undetermined</a>	before March 2004	

Get the RSS feed of the last changes








### Release 0.7.1 - in CVS HEAD




Type	Changes	By
	Log names review so that they both client and server can stay (through links) in the same directory.	<a href="#">carcassi</a>
	NIS update is done every hour and is now thread safe.	<a href="#">carcassi</a>
	Log file permission for the command line tools are set so multiple users can use it (important for admin).	<a href="#">carcassi</a>
	GUMS host can now be used for stress testing and timing the server response.	<a href="#">carcassi</a>
	Added connection pooling on mysql server.	<a href="#">carcassi</a>
	Solved a race condition that would make GUMS hang in some circumstances.	<a href="#">carcassi</a>
	AuthZ callout without GT3, both client and server stubs.	<a href="#">mlorch</a>
	Added code for Privilege Project in GUMS repository and build process.	<a href="#">carcassi</a>

## Release 0.7 - 2005-01-14







Type	Changes	By
	Better logging: server logs all commands with both input and output	<a href="#">carcassi</a>
	More complete command line interface	<a href="#">carcassi</a>
	Web service implementation	<a href="#">carcassi</a>
	HostWildcards can be more than one, comma separated	<a href="#">carcassi</a>
	Support for VOMS Fully Qualified Attribute names	<a href="#">carcassi</a>
	AuthZ service to be contacted by Globus callout	<a href="#">carcassi</a>
	Support for grid3-user-vo- map.txt generation	<a href="#">carcassi</a>
	Many many other refinements...	<a href="#">carcassi</a>

## Release 0.6.1 - 2004-08-10




Type	Changes	By
	Nightly build and reporting with Maven	<a href="#">carcassi</a>
	Removed all the old code from 0.6	<a href="#">carcassi</a>
	Better log system: logs for developer, resource admin and site admin in place	<a href="#">carcassi</a>
	Ability to retrieve groups from within a VOMS server (finally)	<a href="#">carcassi</a>
	No more duplication in the mapfiles	<a href="#">carcassi</a>
	Improved database caching for grid-mapfile: you specify on the server which gatekeeper maps should be generated	<a href="#">carcassi</a>
	Improved error handling (i.e. a failed update on one group doesn't block the others)	<a href="#">carcassi</a>

Type	Changes	By
	Installation through RPMs (cron jobs installed automatically)	<a href="#">carcassi</a>
	Unit tests to Grid3 VO's included	<a href="#">carcassi</a>
	LDAP access improved: can access LCG dev VO	<a href="#">carcassi</a>



## Release 0.6 - 2004-07-01

Type	Changes	By
	XML configuration file for mapping policy	<a href="#">carcassi</a>
	Log infrastructure	<a href="#">carcassi</a>
	More flexible architecture	<a href="#">carcassi</a>
	Decoupled grid-mapfile generation from database caching for distribution on gatekeeper	<a href="#">carcassi</a>
	Web interface to generate grid-mapfiles and map users	<a href="#">carcassi</a>
	Better command line interfaces (feel like Unix commands)	<a href="#">carcassi</a>

## Release 0.5 - 2004-05-20

Type	Changes	By
	GUMS in production at BNL	<a href="#">carcassi</a>
	NISMapper retrieves the GECOS field and matches with certificate CN.	<a href="#">carcassi</a>
	Architecture to allow different type of mappings for different hosts	<a href="#">carcassi</a>

## Release undetermined - before March 2004

Type	Changes	By
	Script to fetch user from VOMS	<a href="#">dtyu</a>
	Script to map user to local account	<a href="#">tomw</a>